UCRL-WEB-200945

MPI Parallelization Problems and Solutions

Table of Contents

Preface

Scope: Drawing on the analysis presented by Aoyama and Nakano in IBM's redbook called "RS/6000 SP Practical MPI Programming" (SG245380), (URL: http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245380.pdf) this document gives a systematic overview, in outline form, of the programming problems encountered on massively parallel distributed-memory computers (such as the IBM SP). For each such problem, the solution(s) offered by, or sometimes required by, the standard MPI (message-passing interface) library are briefly described and compared (especially in light of the circumstances where each solution applies). This document is not intended to be a comprehensive explanation of every MPI problem and solution (that is the role of redbook SG245380), but rather a clear analytic framework for surveying the field of distributed-memory parallel programming and for organizing the many isolated treatments of MPI spread around the World Wide Web (and elsewhere).

Availability: At LLNL, the MPI library is available on all LC production machines. The pathname is /usr/lib/libmpi.a on AIX clusters. This library is in /usr/lib/mpi/lib/libmpi.a on LC's Linux machines.

Consultant: For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, secure e-mail: lc-hotline@pop.llnl.gov).

Printing: The print file for this document can be found at:

```
OCF: http://www.llnl.gov/LCdocs/mpi/mpi.pdf
SCF: https://lc.llnl.gov/LCdocs/mpi/mpi_scf.pdf
```

Introduction

The standard message-passing interface (MPI) library is a way to share data among parallel processes running on distributed-memory massively parallel computers. This summary document provides a logical framework for finding solutions to parallel processing problems, especially those featuring MPI in complex applications. Its (online) table of contents presents a concise outline of those parallelization problems and their MPI solutions.

Every mention of a specific MPI routine here links to that routine's (MPICH) man page (on another LC web server) for easy review of its calling sequence and usage details. One section introduces LLNL's own C-language benchmark suite (Sphinx) for performance testing of MPI on local machines. And a selected bibliography (page 24) at the end summarizes and links to the best free MPI references available online. Future versions will also include (more) solution summaries and links to related MPI material on other sites worldwide organized within the same framework for ease of use.

Unfortunately, IBM's own very useful message-passing manual (redbook) called "RS/6000 SP Practical MPI Programming" (SG245380) (URL: http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245380.pdf) is only available online as one gigantic (238-page) PDF file. So we cannot link directly to its relevant sections and examples from this outline. But this summary document will still grow into a helpful way to organize and find the MPI resources now scattered confusingly around the Internet.

A more fine-grained approach to parallel computing involves creating independent "threads" of execution (POSIX threads or pthreads) *within* one process rather than passing messages among many separate processes. This alternative may be more efficient but is often more complex to program. See the separate local manual called "Pthreads Overview (for LC)" (URL: http://www.llnl.gov/LCdocs/pthreads) for a survey of its known benefits and problems.

The document you are now reading focuses on *designing* good MPI parallel programs. For instructions on how to actually manage, compile, and then execute those parallel programs on the ASCI IBM machines, see the separate POE User Guide (URL: http://www.llnl.gov/LCdocs/poe) (which introduces the key features of IBM's Parallel Operating Environment). For additional advice on efficient I/O for scientific applications at LC and the use of LC's parallel file systems, some of which involves special MPI library routines, see also the I/O Guide for LC. (URL: http://www.llnl.gov/LCdocs/ioguide) A glossary of MPI acronyms (and a few terms) appears as part of the "MPI at LLNL" (URL: http://www.llnl.gov/computing/mpi/index.html) web site, discussed in the Bibliography section at the end of this manual. For an analysis of how best to use the environment variable LD_LIBRARY_PATH on LC's Linux/CHAOS machines to manage access to MPI libraries, especially for nonstandard MPI library versions or MPI use in batch jobs, see the "LD_LIBRARY_PATH Details" section (URL: http://www.llnl.gov/LCdocs/ev/index.jsp?show=s3.5) of LC's Environment Variables user guide. For details on invoking the MPIRUN job-control tool, see MPIRUN for BlueGene/L (URL: http://www.sdsc.edu/user_services/bluegene/docs/mrirun_manual.pdf), a 38-page user guide prepared by IBM's Haifa Research Labs.

Parallelization Effects

Amdahl's Law

If p is the fraction of your program that can be parallelized (and 1-p is the fraction that cannot), and if you run it on n processors, then the ideal parallel running time will be

```
((1-p)  +  p/n)(serial running time)
```

This suggests the importance of carefully idenifying the fraction of your code that can be parallelized, since it sets a limit on improvements in how fast the parallelized program will run.

Effective Bandwidth

The effectiveness of parallelization also depends on how well the program's many processes communicate with each other. Effective bandwith is one way to collectively assess the many factors that influence interprocess communication. See Cherri Pancake's paper "Is Parallelism for You?" in the suggested bibliography (page 24) for an excellent, illustrated online treatment of this issue.

Speedup

Two general strategies promote speedup of your parallelized program:

- decrease the amount of data sent between processes, and
- decrease the number of times that you send the data.

Many sections below spell out specific MPI techniques for carrying out these general speedup strategies.

Parallelization Strategies

The next subsections give a step by step analysis to help systematically pick and deploy parallelization solutions as problems are encountered.

Basic Parallelization Steps

Tune

Tune the serial program, especially the "hot spots" with high CPU use, before you bother to parallelize at all.

Profile

Study the profile of the serial program to begin parallelizing the most CPU-intensive parts (first, and perhaps only those). Two factors, time use and work distribution, should influence your choice of places to parallelize. Assess:

The UNIFORMITY of CPU-time use

NONUNIFORM:
If most CPU time is consumed by a small part of the code, then consider only partial parallelization of the relevant DO loop.
UNIFORM:
If several subloops contribute almost equally to the total run time, then plan to parallelize throughout the whole DO loop.

The likely BALANCE of work among processes

When picking the grain size at which you will parallelize your subroutines, remember the likely effect of your choice on the distribution of work among processes, as this chart shows:

```
                           Distribution of work
                           Balanced       Unbalanced
                           -------------------------
Grain size of             |
subroutines   Coarse|                     Likely
parallelized              |                     (few statements
                          |                     rewritten)
              Fine  |Likely
                          |(many statements
                          |rewritten)
```

Once you start testing an MPI-based parallel version of your code, you can benefit from using mpiP, a "lightweight profiling library" specifically for MPI applications. The mpiP profiler on LC's AIX (IBM) machines:

- has been customized to handle long file names and function names,

- generates much less overhead, less between-task communication, and less total data than most heavy-duty tracing tools, and
- has a local documentation file available on each AIX node at /usr/local/tools/mpiP/README.

Choose Techniques

Determine the most appropriate parallelization techniques for your code:

(A) What kind of distribution (page 14) of data is needed?

(B) What data needs to be sent among processes?

(C) Is it appropriate to "shrink arrays" (page 13) as you parallelize?

(D) Is it more efficient to use PESSL (page 22) routines or to develop your own?

Deploy Techniques

On the basis of this analysis, deploy the parallelization techniques described (later) in this document and elsewhere.

Also keep in mind that you can assist your parallelization by using:

- MODULE statements of Fortran90--to simplify passing arguments during parallelization.
- Incremental parallelization--that is, parallelize loops step by step "from top to bottom" instead of all at once, to make debugging easier when the inevitable mistakes occur.
- Default MALLOC variable settings--
  Before CHAOS 2.0, some users intentionally unset the environment variables MALLOC_MMAP_MAX and MALLOC_TRIM_THRESHOLD on LC's Linux clusters to improve performance. Under current versions of CHAOS, unsetting these variables causes run-time problems, however. So *except on Thunder* you should use the default (preset) MALLOC environment variable values for MPI codes running under Linux.
- MPIRUN with the -env option--
  On BlueGene/L, MPIRUN-executed jobs only have access to those environment variables whose values you explicitly declare with MPIRUN's own -env option. For -env's quoted argument string, use a blank-delimited list of *variable=value* equations.

Troubleshooting

The four typical problems when running a newly parallelized program are that the program:

- does not start,
- ends abnormally,
- starts but becomes inactive, or
- gives the wrong answer.

This section will eventually summarize the likely causes for each such symptom.

Performance Measurements

As of April, 2002, LC's massively parallel IBM computers sometimes showed significant performance problems for MPI programs, especially for those programs that (heavily) use library routines MPI_ALLREDUCE or MPI_BARRIER. One production physics code where MPI_ALLREDUCE was algorithmically expected to use about 1.9% of the total run time, for example, actually spent 90% of its MPI time and 30% of its physics time just executing MPI_ALLREDUCE. Extensive comparative testing by LC staff members suggests that the following kinds of codes are most susceptible to these serious performance problems:

- Parallel codes with fine-grained parallelization,
- Hybrid codes that combine MPI with OpenMP or with POSIX threads (Pthreads), and
- Codes that make heavy use of MPI_ALLREDUCE or MPI_BARRIER.

Users who see (or who wish to avoid) these problems are urged to profile their codes by running /usr/local/mpiP and to read the "IBM Confidential" analysis available at (OCF, special password required, request from the LC Hotline):

`http://www-r.llnl.gov/icc/viewgraphs/viewgraphs02/apr02/jones/index.htm`

for more details and for a few suggestions to work around them. By summer, 2003, IBM and LLNL staff had agreed that latent serial steps deep within AIX parallel operations were a primary cause of these performance problems. For an updated analysis and repair strategy, see

`http://www-r.llnl.gov/icc/viewgraphs/viewgraphs03/aug/jones/jones.pdf`

Parallelizing I/O Operations

Input Cases

For a massively parallel program, there are three ways to handle data input among the many processes:

(1) All processes read the same input file from a shared file system (if there is one).

(2) All processes have a local copy of the input file before computation starts.

(3) One process reads the input file and distributes it to the others using appropriate MPI library routines:

- MPI_BCAST (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Bcast.txt)--used to spread all data to all processes.
- MPI_SCATTER (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Scatter.txt)--used to spread only relevant pieces of data to the processes that need them.

For suggestions on parallel reading from HDF5-format "self-describing" scientific data files, see the "HDF5 Operations" section of the I/O Guide for LC. (URL: http://www.llnl.gov/LCdocs/ioguide) See also the next subsection (on output) for more related suggestions and references.

Output Cases

OUTPUT STRATEGIES.
For a massively parallel program, there are three ways to handle data output from among the many processes:

(1) All processes write to standard output (the default). This generates many relatively small output files, which parallel file systems (see below) distribute among many devices for automatic load balancing.

(2) One process gathers all the data and writes it to a local file. The appropriate MPI library routine for this approach is MPI_GATHER (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Gather.txt).

(3) Each process writes its data sequentially to a shared file. Use routine MPI_BARRIER (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Barrier.txt) to synchronize the processes and avoid data corruption.

For suggestions on parallel writing to HDF5-format "self-describing" scientific data files, see the "HDF5 Operations" section of the I/O Guide for LC. (URL: http://www.llnl.gov/LCdocs/ioguide)

PARALLEL FILE SYSTEMS.
Computer systems with special parallel file-system hardware (such as IBM's General Parallel File System (GPFS) on AIX clusters or the Lustre parallel file system on Linux/CHAOS clusters at LC) enable the use of the MPI-IO parallel I/O interface, an extension to the original MPI library. With MPI-IO, many nodes can not only write at the same time, but they can write to different parts of the same logical file simultaneously. In fact, using MPI-IO to a *standard* globally mounted file system such as /nfs/tmp*n* will actually degrade I/O performance for all users of that file system across all the machines where it is mounted.

For a summary of the positioning, synchronism, and coordination issues involved with using MPI-IO on parallel file systems, see the MPI-IO section of the I/O Guide for LC (URL: http://www.llnl.gov/LCdocs/ioguide). For full technical details on the MPI-IO library routines (and naming conventions), see this specific part of the MPI Forum web site:

`http://www.mpi-forum.org/docs/mpi-20-html/node186.htm`

Note that different parallel file systems interact differently with the MPI-IO library extensions:
(1) For some I/O operations, GPFS outperforms Lustre, while for other operations Lustre performs much better than GPFS does. So expect the unexpected with application code performance if you move from one parallel file system to a different brand, even among LC machines.
(2) Always check the exit codes returned to your application after each read from or write to a parallel file system. Some reveal harmless between-brand behavior differences, while others betray file corruption.
(3) See the section on Lustre's interaction with MPI-IO (URL: http://www.llnl.gov/LCdocs/ioguide/index.jsp?show=s7.4.3) in the I/O Guide for LC for more details. See also the discussion (URL: http://www.llnl.gov/LCdocs/ioguide/index.jsp?show=s7.4.5) in the I/O Guide of how Lustre "stripes" files across multiple storage devices, since striping (which you can somewhat control) interacts with parallel I/O performance.

Parallelizing DO Loops

Distributing Iterations Among Processes

This section describes three solutions to the problem of distributing DO-loop iterations among processes, which then run the iterations in parallel.

Block Distribution

The block distribution approach divides the DO-loop iterations into p equal-sized parts, where p is the number of parallel processes. You can either

- Make the parts as evenly sized as possible, so that all processes get the same number of interations. This allows the use of routine MPI_REDUCE (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Reduce.txt) to manage results afterward.
- Use a specified part size to create the iteration blocks, and leave a remainder (possibly but usually not zero) for one process. The PESSL library uses this method.

Cyclic Distribution

Cyclic distribution assigns DO-loop iterations to parallel processes one iteration at a time, round robin. In some situations (e.g., LU factorization) this can balance the workload better than block distribution, but it can also cause frequent cache misses.

Block-Cyclic Distribution

Block-cyclic distribution assigns DO-loop iterations to parallel processes by first dividing them into equal-sized blocks and then assigning the blocks to processes round robin, cyclicly. The goal is to reduce cache misses yet still get the workload balance of cyclic distribution.

Using Extra (Per-Node) Memory

On a distributed-memory machine (such as the IBM SP), the total amount of memory grows as you add more and more nodes to a computation (not the case on shared-memory computers). You can take advantage of the extra per-node memory you add every time you add a process by either

- Keeping the same problem resolution and computing more data, or
- Keeping the same data and increasing the problem resolution.

The second approach is often very beneficial on practical simulation problems, where you can either

- Shrink arrays--let each process use all of its memory on just a portion of the original array for a finer resolution, or
- Use the ALLOCATE statement to implicitly assign space at run time if you do not know the size of the array or the total number of processes before you run the program.

Parallelizing Nested Loops

You can parallelize nested DO-loops in a way that minimizes the communication between processes as well as the frequency of cache misses if you consider:

- the storage order for multidimensional arrays. Fortran stores such arrays in column-major order, but C stores them in row-major order.
- the dependence of each element on its neighboring elements in the same row.
- the possible dependence of an element on its neighbors in more than just one dimension too.

Message Passing

Message passing is the solution to the (distributed-memory) problem of processes not having the data they need once computational iterations have been distributed among them. Two basic cases occur (addressed in the subsections below):

- No order ("loop-carried") dependencies exist among the iterations.
- Order ("loop-carried") dependencies do exist among the iterations that are distributed.

No Order Dependencies

Broadcast of a Single Element

You can broadcast a single crucial data element to all processes in a massively parallel program by using the MPI library routine called MPI_BCAST (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Bcast.txt).

1-D Finite Difference Method Parallelized

You can parallelize the one-dimensional finite difference method by using MPI_ISEND (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Isend.txt) and MPI_IRECV (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Irecv.txt) to systematically transmit data on the boundary of a one-dimensional array.

Note that the "I routines" MPI_ISEND (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Isend.txt) and MPI_IRECV (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Irecv.txt) often give better performance on IBM machines than the standard versions MPI_SEND (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Send.txt) and MPI_RECV (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Recv.txt), but they sacrifice portability to nonIBM platforms.

Bulk Data Transmissions

Gathering Data to One Process

Gathering data to one process from other processes usually occurs near the end of a parallel program. There are four distinct cases here that differ in their most appropriate message-passing routines and in how those routines are applied, as this table shows:

```
                          Data location in memory
                     Contiguous          Noncontiguous
Send and             ---------------------------------
recieve buffers:    |
                    |
     NO overlap     | (1) Use            (3) Trivial extension
                    | MPI_GATHERV        of case (4) below
                    |
     DO overlap     | (2) Point-to-point (4) Use
                    | communication:     ALLOCATE, MPI_ISEND,
                    | ALLOCATE,          MPI_RECV, MPI_WAIT
                    | MPI_IRECV,
                    | MPI_WAIT
```

Note that the "I routines" MPI_ISEND (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Isend.txt) and MPI_IRECV (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Irecv.txt) often give better performance on IBM machines than the standard versions MPI_SEND (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Send.txt) and MPI_RECV (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Recv.txt), but they sacrifice portability to nonIBM platforms.

Synchronizing Data

Synchronizing data means making sure that all processes have up-to-date data at the same time, even part way through a parallel computation. There are four distinct cases here that differ in their most appropriate message-passing routines and in how those routines are applied, as this table shows:

```
                          Data location in memory
                     Contiguous          Noncontiguous
Send and             ---------------------------------
recieve buffers:    |
                    |
     NO overlap     | (1) Use            [not discussed]
                    | MPI_ALLGATHERV
                    |
     DO overlap     | (2) Use ALLOCATE,  [not discussed]
                    | MPI_BCAST (or
                    | MPI_IBCAST) to
                    | broadcast as many
                    | times as there are
                    | processes
```

Note that the "I routine" MPI_IBCAST (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Ibcast.txt) often gives better performance on IBM machines than the standard version MPI_BCAST (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Bcast.txt), but it sacrifices portability to nonIBM platforms.

Transposing Block Distributions

This technique involves changing the distribution of a matrix (from column-wise block to row-wise block) during a parallel computation. You need to use MPI_SEND (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Send.txt) and MPI_RECV (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Recv.txt) along with derived data types for each block.

Reduction Operations

General Reduction

The goal here is to "gather data distributed over [many] processes and do some computation on the way." Two MPI routines are most relevant:

MPI_ALLREDUCE

is best when *every* process needs the value of a sum. (See the comments on IBM performance problems when using MPI_ALLREDUCE, in an earlier section. (page 9))

MPI_REDUCE

is best when *only one* process needs the value of a sum.

Superposition

Superposition is a way to use MPI_REDUCE (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Reduce.txt) to gather data and to avoid derived data types or data packing (but at the cost of sending more than the minimum number of transmissions between processes).

Order Dependencies

Nested Loops

When loops are nested, there are two ways to parallelize order (loop-carried) dependencies: the pipeline method and twisted decomposition.

Pipeline Method

The pipeline method is the way to parallelize a loop that has a flow dependence, so that each iteration has to be executed strictly in order. The Incomplete Cholesky Conjugate Gradient Method is an example of such a situation. Here there is no danger of deadlock.

Twisted Decomposition

This is the way to parallelize when one loop is flow dependent on one dimension of a matrix, and a second loop is simultaneously flow dependent on the second dimension of the matrix. Here, unlike with the pipeline method, there is a danger of deadlock.

Nonnested Loops

When loops are not nested, parallelization requires a different approach than when they are nested.

Prefix Sum Method

This parallelization technique is suitable for nonnested loops spread over a large number of processes (it is inefficient when used with only a small number of processes). It involves the MPI routine MPI_SCAN (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Scan.txt), and you may extend it to other data types and operations on them by invoking MPI_OP_CREATE (URL: http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Op_create.txt).

Advanced MPI Programming

This section covers parallel-programming situations that are more complex, and that resemble typical application programs in the elaborateness of the dependencies they include.

2-D Finite Difference Method

The two-dimensional finite difference method typically involves data dependencies in both dimensions of an array. Three ways of distributing iterations through the array are possible:

- Column-wise block distribution.
- Row-wise block distribution.
- Block distribution in both dimensions, which can be either
  (1) simple, or
  (2) complex.

Finite Element Method

The finite element method often involves a more irregular data dependence that other methods, so it is harder to parallelize successfully. Also, on irregular meshes, the number of assigned nodes may be poorly balanced among processes if you divide the data evenly in terms of elements.

LU (Lower-Upper) Factorization

LU (lower-upper) factorization solves linear equations with dense matrices by factoring the matrix into a lower triangular and an upper triangular matrix. IBM's Parallel Engineering and Scientific Subroutine Library (PESSL) contains subroutines for LU factorization that usually outperform hand-coded solutions. So you should study the PESSL routines and their section below (page 22) when parallelizing programs that involve LU factorization.

SOR (Successive Over-Relaxation) Method

The successive over-relaxation (SOR) method solves two-dimensional Laplace equations. When parallelizing such programs, you must distinguish among three three cases:

- Red-black SOR, which differently "colors" every other element and then updates the red elements and the black elements alternately.
- Zebra SOR, which colors whole columns alternately, like a zebra's stripes.
- Four-color SOR, which is a generalization of reb-black SOR to handle the case when more neighbors are involved in updating each element.

Monte Carlo Method

The Monte Carlo method reveals the special problems that arise when you try to parallelize random number generation and use.

Molecular Dynamics

Molecular dynamics involves the "distinct element method," in which only a few of the inner loops use most of the CPU time. So your parallelization efforts should focus on just those CPU-intensive loops.

MPMD Models

Most of this document's sections apply to SPMD (single program multiple data) parallel programming. But MPI techniques can also handle cases of MPMD (multiple program multiple data), where different programs run in parallel and communicate with each other. You toggle between theses cases by using the environment variable MPI_PGMMODEL, whose value can be spmd (the default) or mpmd.

The "master/worker" approach, where one process (the master) coordinates all the others (the workers) is the basic way to deal with MPMD.

Parallel ESSL (Dropped)

[This library is no longer supported by IBM documentation.]

Multifrontal Method

The multifrontal method is a very efficient way to solve sparse symmetric matrices with wide bands. It is NOT, however, supported by IBM's Engineering and Scientific Subroutine Library (ESSL) or its parallelized counterpart (PESSL). For a robust, easy to use, and high-performance set of multifrontal subroutines, you should instead rely on the Watson Symmetric Sparse Matrix Package (WSSMP). WSSMP documentation is available online at

`http://www.research.ibm.com/mathsci/ams/ams_WSSMP.htm`

WSSMP works either with threads on a shared-memory machine or as a scalable parallel solver in a message-passing environment (such as the IBM SP).

MPI Performance Benchmarks

LLNL's Center for Applied Scientific Computing (CASC) now provides as publicly downloadable code a C-language integrated parallel microbenchmark suite to conduct performance tests of MPI on many different platforms (Compaq, IBM, SGI, and Sun). LLNL's MPI benchmark suite, called Sphinx, has these features:

- Accesses each test action (such as message pingpong) through a function pointer, allowing different threads or tasks to execute different functions at once. This supports measurement of highly complex parallel actions.
- Times repeated calls (iterations) of each test action, stopping either when the standard deviation of the repetitions becomes less than a user-specified percentage of their mean or, if that never happens, after a user-specified maximum number of repetitions.
- (Optionally) corrects for test-suite ("harness") overhead and automatically warns if that overhead exceeds the measurement value of the test.
- Is highly portable (vendor-dependent binding of POSIX threads to processors is the chief threat to Sphinx portability).
- Covers (with suitable adaptations) pthreads, MPI, and OpenMP performance testing. Documentation at the Sphinx web site (below) specifically explains which tests apply to which features of the three approaches to parallelization.

Sphinx is available to the public at this open URL:

`http://www.llnl.gov/CASC/sphinx`

This Sphinx web site (UCLR-CODE-99026) provides all needed usage information and relevant files, including:

- A descriptive inventory of every file in the current Sphinx distribution, which includes every input file for the ASCI milepost tests.
- Build and execution instructions for the test suite.
- Input file format and the input modes that Sphinx accepts.
- Output file format and the four Sphinx output streams.
- Specific test descriptions and allowed independent variables.
- References to (and in some cases even the full text of) published papers that present and discuss Sphinx results.

MPI Online Bibliography

This section critically summarizes, gives the URLs for, and interactively links to the best free online information sources (some tutorial, some reference) about parallelization in general and MPI in particular, culled from many unhelpful alternatives.

Cherri M. Pancake (Oregon State University)
Is Parallelism for You? Rules of Thumb for Computational Scientists and Engineers
http://web.engr.oregonstate.edu/~pancake/papers/IsParall.html (URL: http://web.engr.oregonstate.edu/~pancake/papers/IsParall.html)
19 pages (HTML)

This is a basic, very lucid, and quite specific analysis of alternative aproaches to parallelization, aimed at helping a scientist or engineer answer the title question about any planned parallel programming (or conversion) project. Pancake offers 16 explicit rules of thumb for when and how to parallellize, based on her systematic, illustrated comparison of:

- application characteristics: perfect or pipeline parallelism, fully or loosely synchronous applications.
- control models: single (SIMD) or multiple (MIMD) instructions for multiple data.
- memory models: shared memory, distributed memory, or shared-memory CPU groups combined into larger distributed-memory machines (symmetric multiprocessors, SMPs).
- programming models: single or multiple executables participating in the parallel run.

Says Pancake, "the purpose and nature of your application are the most important indications of how successful parallelization will be. Your choice of parallel computer and plan of attack will have significant impact too, not just on performance but also [on] the level-of-effort required....the techniques I present for estimating likely performance gains are drawn from the experiences of hundreds of computational scientists and engineers at national labs, universities, and research facilities."

Peter S. Pacheco (Univ. of San Francisco)
A User's Guide to MPI
ftp://math.usfca.edu/pub/MPI/mpi/guide.ps
51 pages (PostScript and PDF available)

Pacheco subsequently published a professional textbook called "Parallel Programming with MPI" (San Francisco: Morgan Kaufmann, 418 pages, 1997), and this MPI guide is his freeware draft for that project (at an anonymous FTP site). He describes it as "a brief tutorial introduction to some of the more important features of MPI," and it contains C programming examples exclusively. Pacheco introduces the key MPI library routines gradually throughout the text as he tours the main issues in parallel programming, including

- point-to-point communication (Send, Recv),
- collective communication (Bcast, Reduce),
- grouping exchanged data to minimize message traffic, and
- using grids and graphs to manage multiple MPI "communicators."

The analysis here is very clear but not very complete. The many examples all show how MPI routines behave (in practice cases), but not how you might typically deploy them in large and complex application codes.

A Fortran version of this guide is now available from Hong Kong University, but only in MS Word 7.0 (word processing) format, at this URL:
http://www.hku.hk/cc/sp2/ftp/mpi/MPI_ug_in_FORTRAN.doc (URL:
http://www.hku.hk/cc/sp2/ftp/mpi/MPI_ug_in_FORTRAN.doc)

Yukiya Aoyama
Jun Nakano
RS/6000 SP: Practical MPI Programming
http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245380.pdf (URL: http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245380.pdf)
238 pages (PDF only)

This hefty manual "helps you write MPI (Message Passing Interface) programs that run on distributed memory machines such as the RS/6000 SP," and thus it specifically addresses LC's AIX production machines. It covers both C and Fortran codes (most examples are Fortran), and a 45-page appendix conveniently reproduces example-enhanced MAN pages for every MPI library routine.

Aoyama and Nakano, on the staff at IBM Japan, have in this "redbook" (SG245380) achieved the often conflicting goals of clear explanation and technically thorough treatment. Their introductory conceptual chapters cover the same background topics as does Pacheco, only with more detail, more diagrams, and more insight into the MPI library.

The bulk of the book deals with parallelization strategies, message passing as a solution to typical communication problems, and advanced MPI programming (in code with complex dependencies). LC's own MPI Parallelization Guide reflects the rich content and crisp structure of SG245380; it is a distillation of that larger book's educational agenda (so we won't repeat it here). Aoyama and Nakano never flinch at examining the intricate MPI problems posed by actual application programs, yet their cogent analysis and often ingeniously clear illustrations keep the discussion effective and revealing.

This is certainly the most elaborate as well as the most helpful MPI manual now freely available online. Lapses in English translation are rare and never serious (see also the next, more specialized, item).

Gary Mullen-Schultz
BlueGene/L: Application Development
http://www.redbooks.ibm.com/abstracts/sg247179.html (URL: http://www.redbooks.ibm.com/abstracts/sg247179.html)
172 pages (HTML and PDF versions of the same text)

Although this manual also covers control system APIs and performance analysis, its largest part (Part I, 68 pages plus an appendix) addresses MPI application support on the unusual BlueGene/L architecture. Chapters here discuss MPI code management on BlueGene/L more than strategic program design. Topics include memory and link issues, coprocessor versus virtual-node mode, system calls, compiling and tuning on BlueGene/L, running and debugging, and checkpointing to restart. From a fork at the (above) abstract, both HTML and PDF versions of this book are available online.

Note that this IBM text assumes a BlueGene/L machine using LoadLeveler as the underlying job manager. At LC, BlueGene/L uses instead the locally developed resource manager SLURM (URL: http://www.llnl.gov/LCdocs/slurm) because the machine runs Linux/CHAOS rather than AIX as its operating system.

Appendix D of this manual explains the role of four environment variables relevant to MPI codes but unique to BlueGene/L:
BGLMPI_COLLECTIVE_DISABLE
BGLMPI_EAGER
BGLMPI_RVZ
BGLMPI_RZV

Blaise Barney
Message Passing Interface (ASC Support Training)
http://www.llnl.gov/computing/tutorials/mpi (URL: http://www.llnl.gov/computing/tutorials/mpi)
50 pages (HTML)

After 10 pages of background on "getting started with the message passing paradigm," this becomes a systematic explanatory tour of the MPI library. Unlike Pacheco, Aoyama, and Nakano, this is not a problem-oriented treatment of how to apply MPI to typical application-code challenges. Rather it is a careful review of what programming resources the MPI library offers to the prospective parallel programmer, organized by broad categories of routine.

For example, the "Environment Management" section gives the Fortran and C calling sequences for all nine MPI routines (such as MPI_Init) in this category, with brief comparative comments on the role of each. Typical uses appear in dozen-line Fortran and C code examples at the end of each section. Supplementary tables cover MPI data types and MPI routine names alphabetically (always in functional categories), where each name links to the corresponding MAN page for usage details. Also introduced here are the extensions to the original MPI specifications found in MPI-2.

Barney's approach to MPI is very complete (as a library survey) and well balanced between C and Fortran (unlike Pacheco). But it leaves complex application problems largely to the user (unlike the examples of Aoyama and Nakano).

Argonne National Laboratory
Message Passing Interface (MPI) Standard (site)
http://www-unix.mcs.anl.gov/mpi (URL: http://www-unix.mcs.anl.gov/mpi)
HTML with many links

Many web sites claim to provide MPI information and to gather useful MPI links. Most, however, point to one another circularly, have poor annotation for their links, and add local material that is too meaningless in isolation, too incomplete, or too disorganized to justify the time spent finding and reading it. Argonne National Laboratory's (ANL) Mathematics and Computer Science Division stands apart, with a genuinely useful, thorough, and fairly well organized collection of MPI postings and links. Other sites will eventually lead you to ANL, so starting here will save hours of study time.

ANL's annotations spell out, especially below the top level of MPI topics, just how many relevant resources are assembled here:

- Research papers on MPI,
- MPI software tools,
- MPI libraries,
- Tutorials and texts, including those summarized in this section (except the IBM redbook),
- The MPI official standard and FAQ, for completeness.

MPI users who want to bookmark one truly helpful "jump page" for future searches should find that this one meet their needs better than most. (Note that even here, however, the quality of linked information varies greatly.)

LC Hotline Staff
MPI Use Summary
/usr/local/docs/MPI_Use_Summary
200-300 lines of plain text

On each LC IBM and Compaq massively parallel machine, the LC Hotline staff maintains a simple 200-to-300-line file that gives very current and very localized information on:

- The location of the executable compiler scripts available on that specific machine,
- The current correlation between script names (mpicc, old_mpicc, new_mpicc) and the compiler version (MPICH-1.1.2, etc.) that each named script invokes,
- Hints about compiler options especially relevant to MPI programs on that machine, as well as about the most important local MPI environment variables,
- A warning that MPICH is not thread safe, so its output must be single threaded,
- Short but explicit examples of how to locally compile C, C++ (various brands), Fortran77, and Fortran90 with MPICH, and
- A brief comparison of MPICH with the native MPI compiler(s) on each specific machine.

Consulting this local help file along with LC's POE User Guide (URL: http://www.llnl.gov/LCdocs/poe) provides solid practical context for managing, compiling, and executing the MPI programs whose design is discussed in the manual that you are reading.

Terry Jones, LLNL
MPI at LLNL
http://www.llnl.gov/computing/mpi/index.html (URL: http://www.llnl.gov/computing/mpi/index.html)
15 pages (HTML)

This brief summary of MPI resources at LC begins with an overview of LLNL industrial collaboration on MPI, on LLNL's role in the MPI specification, and on the names and addresses of half a dozen local MPI contacts (for A and B divisions, for IBM problem resolution, etc.). Also here are announcements of future MPI seminars at LLNL ("News and Events"), links to external resources, and MPI library loading information for IBM, Compaq, and Quadrics-switch (Linux) environments. Some of the same online reference material cited in this manual is cited on the "MPI at LLNL" pages too.

Because MPI sometimes interacts with other system features in unexpected ways, three other sections of "MPI at LLNL" are quite helpful for problem debugging:

- Open Issues, Gotchas, and Recent Changes.
  This subset of the site discusses how to deal with such issues as unexpected application hangs, switch bugs that affect MPI subroutine calls, or whether or not to invoke "large (memory) pages" under AIX.
- Glossary.
  This section offers short definitions in alphabetical order of acronyms (and some technical terms) that commonly but mysteriously appear in MPI discussions, from AMPI to VP.
- Environment Variables.
  This section explains 21 LIBELAN_ environment variables that help MPI codes running on LC Linux machines to make effective use of the Quadrics switch (for better performance and debugging) through the Elan Communication Library. You may also want to see the general comparison of locally used and MPI-relevant environment variables (on both AIX and Linux/CHAOS systems) in LC's Environment Variables user manual (URL: http://www.llnl.gov/LCdocs/ev).

Jeffery Vetter, Chris Chambreau, LLNL
MPIP, MPIPVIEW (MPI Profiling)
http://www.llnl.gov/CASC/mpip (URL: http://www.llnl.gov/CASC/mpip)
12 pages (HTML)

MPIP (usually cited in reverse case as mpiP) is a locally developed and iteratively refined profiling library specifically designed for MPI application programs. It generates much less overhead and data than standard tracing tools, and it uses communication only to generate its output report. MPIP works under Linux (including BlueGene/L), Tru64, and AIX systems. The output GUI, called MPIPVIEW, has even been optimized to support display of the huge output files made on BlueGene/L. The project's very concise and well-organized explanatory website (URL listed above) offers:

- downloads of the latest version of MPIP,
- very explicit linking instructions, both generally and with specific examples for each LLNL platform (combination of operating system and compiler),
- an explanatory summary of the available MPIP configuration options (currently about a dozen),
- an index of MPI routines that MPIP profiles (sorted alphabetically),
- an index of MPI routines for which MPIP gathers sent-message size data, and
- an index of MPI routines for which MPIP gathers I/O data.

Disclaimer

Keyword Index

To see an alphabetical list of keywords for this document, consult the next section (page 36).

Alphabetical List of Keywords

Date and Revisions

Revision Date	Keyword Affected	Description of Change
26Jun06	introduction	Cross ref to MPIRUN manual added.
	deploy-techniques	Cross ref to MPIRUN manual added.
03May06	references	MPIP, MPIPVIEW subsection added.
22Feb06	introduction	LD_LIBRARY_PATH cross ref added.
	references	Cross ref to Env Var manual added.
14Nov05	pessl	Library support dropped by IBM.
	references	BlueGene/L MPI reference added.
	index	PESSL entry deleted.
27Sep05	i-o-output	More MPI-IO/Lustre interaction details.
	references	Details updated.
09Mar05	deploy-techniques	Warning on unsetting MALLOC variables.
	references	LIBELAN_ ent. vars. cross referenced.
16Feb05	i-o-output	MPI-IO and Lustre interaction warning.
10Jan05	references	Problems, glossary added to "MPI at LLNL."
08Mar04	references	Several MPI URLs updated.
02Sep03	performance	Revised analysis of AIX performance probs.
	profile	Support for mpiP profiler added.
20Aug02	i-o-input	Another cross ref. added.
	i-o-output	MPI-IO discussion added.
	introduction	Cross ref to I/O Guide added.
25Jul02	i-o-input	Cross ref on parallel input from HDF5.
	i-o-output	Cross ref on parallel output to HDF5.
	references	Local MPI contact page added.
06May02	performance	MPI_ALLREDUCE performance problem noted.
	reduce-data	Cross reference added to problem.
15Oct01	introduction	Cross ref to POE Guide added.
	references	Local MPI help file added.
11Jun01	benchmarks	New section on Sphinx benchmarks.
	index	New keyword for new section.
20Feb01	introduction	Cross ref added to LC's Pthreads manual.
14Aug00	references	IBM site now requires registration.
	scope	New printing instructions.
10Nov99	references	Best MPI references added.

	entire	All MPI routines linked to MAN pages.
25Oct99	entire	First draft edition of this document.

TRG (26Jun06)

UCRL-WEB-200945

TRG (26Jun06) Contact on the OCF: lc-hotline@llnl.gov, on the SCF: lc-hotline@pop.llnl.gov